Team Gao: Mobile Operating Vehicle Electronics (SDP 2009)

Shelby Berleus, Lukasz Gawel, Karan Mendiratta, Juan Peralta

Abstract

We've built a wireless network-car interface unit to obtain information about car dynamics over a Bluetooth PAN wireless RF network, like, speed, RPMs, engine air-injection level , manifold air pressure etc. In addition, a programmable GPS-chip and a piece of software embedded with the system will provide insightful analysis of for example, the most fuel- efficient route to a particular destination, based on the information retrieved from the car.

I. Background

In today's oil driven economy, a major rises in gas prices can cause calamity which is one reason for using fuel judiciously. It is estimated that earth will run out of non-renewable fuel by the beginning of the next century. In response to the undesirable facts many individuals are looking to purchase cars with fuel efficiency in mind. Currently there are car companies that are producing hybrid vehicles which improve fuel economy; however the cost to buy and new hybrid car may be prohibitive to many. To help consumers who have older cars or have economic barriers we have a solution that will help improve fuel economy.

This is why we propose the Mobile Operational Vehicle Electronics (M.O.V.E) to help those in need of improving their car fuel economy. This system will give users real time data about his car such as the fuel level, the number or RPM's. Data from your car will be analyzed to help each driver optimize their fuel consumption and will give suggestion on how to change or modify one's driving patterns to save gas. The aim of our project is to help people save money on their gas consumption and give them the ability to make changes to their driving style if deemed necessary.

II. DESIGN OVERVIEW

- A microcontroller (Arduino Board) is used to interface the control circuitry to the OBD (On Board Diagnostics) Board Diagnostics) port. The microcontroller requests data from the OBD port at programmed intervals.
- The data is stored on the SD card comprises the GPS co- ordinates, and the OBD data, and transmitted using the XBee module on our circuit board to the user computer with another receiver Xbee connected to it.
- The data transmitted over the Bluetooth PAN is stored in a text file created using a code written in C#.
- The user computer that receives the data packet from the car talks to the microcontroller via a serial interface.
- The software program reads the contents of the text file, maps out the route the user took using the GPS co-ordinates on google maps and uses the OBD data to perform fuel analysis and presents it to the user.
- The microcontroller samples the data from the GPS module every second.

2. The device will relay updates to the system whenever the receiver Xbee is in range of the transmitter Xbee in the car.

3. The system will be interfaced via Bluetooth PAN network (protocol 802.15.4)

4. Role of voltage regulators:
   - Car battery has standard 12 V DC.
   - Arduino board operates on standard 5V and 0V.
   - Xbee modules operate on 3.3V DC.
   - Based on the above voltage requirements of different components, we set up different voltage regulators in different parts of our control circuitry to regulate the voltage as per component requirement.

5. The microcontroller being used for this prototype is Arduino Board (ATmega8).

III. DELIVERABLES

The goal is to have a device successfully communicating with the vehicle via Bluetooth PAN wireless network and have a working prototype of our system ready.

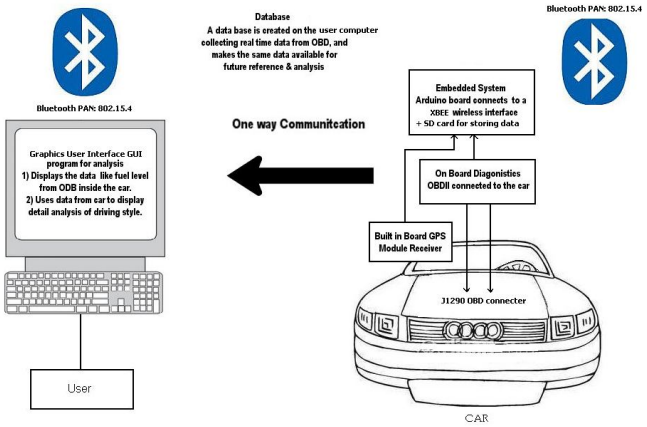
The following are the deliverables we have at the end of the project:

- Working communication between vehicle and ELM 322 chip and SD card - user computer.
- Wireless transmission of GPS positioning and OBD data stored on the memory card to user computer using the Xbee module(s).
- GUI for the end user interface.
- Insightful analysis of fuel consumption pattern of the user, in comparison with standard database of fuel-consumption patterns as released by US government.
- Recommendations of ways to improve fuel efficiency including laying out best routes on google maps.
- Block Diagram with high level view of M.O.V.E
- User Manual

IV. USER INTERFACE

The user interface consists of a GUI giving the user an array of different options like viewing route using GPS data (google maps API), view daily/weekly/monthly fuel analysis reports, to chose their own car from a drop down menu, for a better, more comprehensive analysis.

V. BLOCK DIAGRAM


Bluetooth PAN: 802.15.4
Graphics User Interface GUI program for analysis
1) Displays the data like fuel level from ODB inside the car.
2) Uses data from car to display detail analysis of driving style.
User
Database
A data base is created on the user computer collecting real time data from OBD, and makes the same data available for future reference & analysis
One way Communitcation
Bluetooth PAN: 802.15.4
Embedded System
Arduino board connects to a XBEE wireless interface + SD card for storing data
On Board Diagonistics OBDII connected to the car
Built in Board GPS Module Receiver
J1290 OBD connecter
CAR

VI. INPUT

The inputs to M.O.V.E come from the GPS module and the OBDII connection to the car, stored on the SD card.

The requested data will be transmitted to the user computer and stored in a local database when the receiver-transmitter Xbee modules are in range.

The microcontroller on the control circuitry reads file contents on the SD card and transmits that character by character, Arduino processing on the user end puts these received characters in a text file, which is then read by the software program.

VII. OUTPUT

Output will be desired function of the received request from the end user, which includes:

- fuel consumption pattern,
- analysis of fuel consumption over a period of time,
- route mapping, with most fuel efficient route recommendation,
- Driving style inferences.

VIII. USER MANUAL

- M.O.V.E. users will log into user GUI by clicking on the link provided by our software that will allow them access to M.O.V.E. homepage and monitor their fuel economy.
- Once users are authenticated then an M.O.V.E. user will able to perform many different tasks as needed.
    - E.g. once logged in a user might desire to view his fuel economy over the past month. Since all the data is constantly stored and updated on a database setup on the user computer, the software we write pull out the required data out of the data make, draws meaningful analysis and presents it to the user.
- A user will also be able to navigate and map out the routes he takes using google maps API we've created in our software program.
- Once the user is finished with all tasks a user may end their session by selected the exit icon/link that will exit them for their M.O.V.E main page.

VIII. FINAL SPECIFICATIONS

1. User Interface

A fully developed GUI user interface that enlists different options for analysis and viewing google maps route. Once a task is performed, the software jumps to the home page with different options

2. Database

Implemented in SQL server, stores all the data wireless received from the car.

3. Wireless communication using the Xbee modules that adhere to the 802.15.4 wireless networking protocol. (Bluetooth PAN)

4. Functional circuit board placed in the car comprising GPS module, SD-card, RS-232 interface and ELM322 circuitry.

IX. PROJECT MANAGEMENT: DIVISION OF TASKS

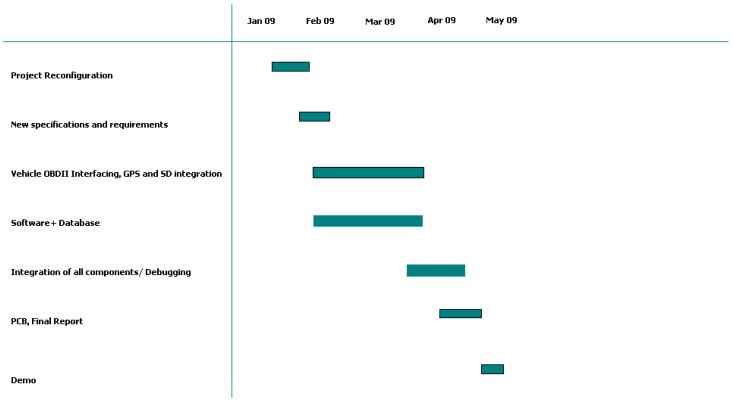
Karan Mendiratta Shelby Berleus Lukasz Gawel	Vehicle Interfacing, GPS, SD-card integration, XBee setup
Juan Peralta	Software+ Database

X. COST ANALYSIS

GPS Module: $60
XBee Modules: $35
OBDII cable: $10
ELM322 chip: $15
Misc: $5

Total prototyping cost = $ 125

XI. GANTT CHART



XII. COMPONENT DESCRIPTION

GPS Module/SD Card

The purpose of the GPS is to receive accurate real time location coordinates of a vehicle. The GPS should in the most optimal conditions. That means the GPS must be used outside with a clear view of the sky in order to fix to satellites. With an unobstructed view of the sky the GPS module should work anywhere in the world twenty-four hours a day. Once the GPS is fixed to satellites then GPS sentences can be obtained. These sentences will be stored in a SD Card that will hold real time data of time, location coordinates, and altitude.

Operation of GPS Module/SD Card

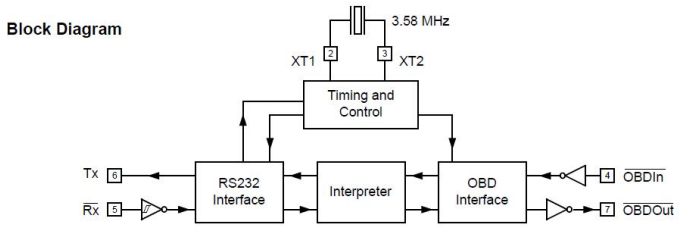
This system uses a Parallax GPS Receiver Module and a SD card that is connected to an Arduino board. The GPS module will send data to the Arduino every second and in turn the Arduino will then take that data and send it to the SD card for data logging. Data will move in this direction until one round trip is made and the car return home. It is at this point that data will be read by the Arduino so it can be wirelessly transmitted by an X-Bee module; that is also connected to the Arduino, to the end user computer. This will be one self contained that sit on either the car dashboard, or car roof top for good view of the sky. After the GPS is position in either location then the power must be turned on in order for the GPS to fix to satellites. When the GPS has a solid red light the GPS has established a connection to satellites and is prepared for logging. When the GPS is connected it will begin logging and storing GPS sentences.

GPS Sentences

The Parallax GPS provides standard, raw NMEA0183 (National Marine Electronics Association) strings or specific user-requested data via the serial command interface, tracking of up to 12 satellites, and WAAS/EGNOS (Wide Area Augmentation System/European Geostationary Navigation Overlay Service) functionality for more accurate positioning results. The NMEA0183 is provided as a series of comma-delimited ASCII strings, each preceded with an identifying header. For example $GPGSV, $GPGGA, $GPGSA, and $GPRMC. The data is transmitted as a 4800bps string of 8-bit ASCII characters. Thus, any microcontroller with a serial port can extract data from a GPS module. But, modules do not produce “plain text” location information. Instead, they create standardized “sentences,” such as: $GPGGA,170834,4124.8963,N,08151.6838,W,1,05,1.5,280.2,M,-34.0,M,,,*75. For the purposes of this project the most important GPS header would be the $GPRMC which has most important data that is need such as time, date, latitude, longitude, speed, and altitude. So all other sentences will be filtered out except for the $GPRMC header.

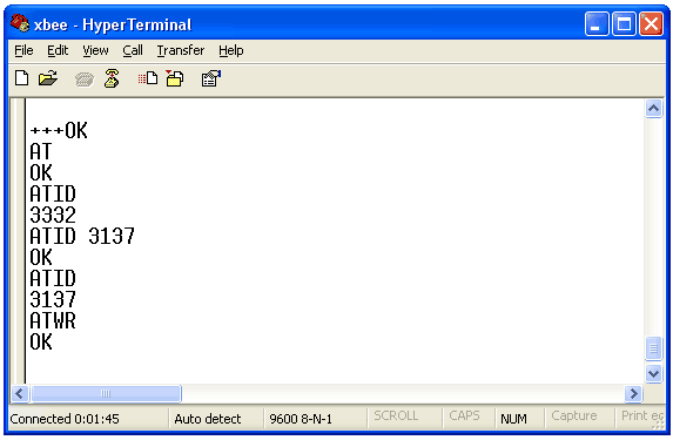
ELM322- OBDII IC

The ELM322 is an 8 pin integrated circuit that is able to change the data rate and reformat the OBD signals into easily recognizable ASCII characters. This allows virtually any personal computer to communicate with an OBD equipped vehicle using a standard serial port and a terminal program.



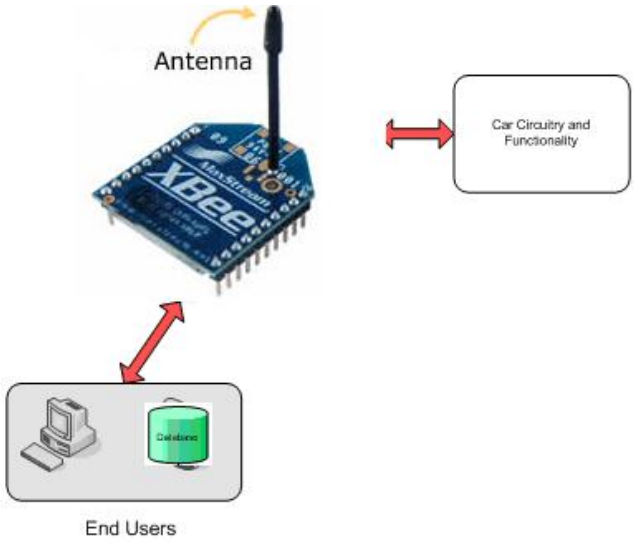
XIII. Configuring XBee

We use the most basic way to communicate using the XBee modems, i.e., point to point. That means one modem communicating with another modem. Serial data goes in one XBee, is transmitted wirelessly, and goes out the other & vice versa.
We set the network ID (the PAN - Personal Area Nework - ID) to a unique value. By default all XBee's use PAN ID #3332.
Connecting to Arduino: We set up the PAN ID and baud rate for the two modems. We chose the baud rate as 9600.
We then setup a point-to-point 'tunnel' between the two XBees. When we type a string into the terminal at the computer connected to the transmitter, the same string will appear in the Arduino's Serial Monitor as the receiver computer upon successful connection.



XIII. USER GUI

- Current car systems on the market today, all have range limitations.
- Through M.O.V.E, range limitations have been eliminated by using long range Xbee modules (upto ½ mile) to communicate with the car’s onboard device.



- The above diagram shows a high level view of the communication system between the end user, database, and vehicle being used.
- The database server, SQL Server 2005, plays an integral role for storing member data and also all capable functions. By utilizing a database it allows for growth as there would no code changes. This took careful planning of tables to allow for such growth. When planning for a database that will continue to grow and expand it is extremely important to take this into consideration in this way there will not be a need to redesign an entire database scheme.

- Tables and queries were created in order to pull data depending on function selected by the user logged in.
- The program handles all incoming communication from the XBee transmitter in the cae and translates this into commands that must be performed once engaged.
- A google map API is also incorporated into our software program that maps out the route the user took using the received GPS co-ordinates.

When all of this is put together we have a fully functional application that delivers fuel-analysis functionality to a user anywhere on the go.

XIV: Result

Specifications met:

- Established a working link between the host (car) and the client (user).
- The host is able to log in on a fully developed, secure GUI interface with numerous customizable options.
- Fully developed GUI web interface
- Client computer-car wireless communication. (Bluetooth Pan)
- Sampling and storing GPS and OBD data on memory card transmitting it to receiver Xbee when in range.
- Prototype of the project.

XV. APPENDIX

[1] Microcontroller Code

Xbee-Arduino code:

```
#include <NewSoftSerial.h>

NewSoftSerial mySerial =  NewSoftSerial(2, 3);


void setup()  {
  pinMode(13, OUTPUT);
  Serial.begin(9600);
  Serial.println("Goodnight moon!");
  // set the data rate for the SoftwareSerial port
  mySerial.begin(9600);
  mySerial.println("Hello, world?");
}



void loop()                     // run over and over again
{

  if (mySerial.available()) {
      Serial.print((char)mySerial.read());
  }
  if (Serial.available()) {
      mySerial.print((char)Serial.read());
  }
  delay(100);
}
```

OBD+GPS Code:

```
#include <Wire.h>
#include <NewSoftSerial.h>
#include <WProgram.h>
#include <avr/pgmspace.h>
#include <microfat2.h>
#include <mmc.h>
#define rxPin 0
#define txPin 1
char incomingByte = 1;
char gpsByte = 1;
char data[8];
char sp[5];
char rm[5];
char im[5];
char it[5];
char info[99];
char finish[99];
int p = 0;
int i = 0;
int readok = 0;
NewSoftSerial gps(2, 3);

void setup() {
      Serial.begin(9600);
      gps.begin(4800);
      Wire.begin();
      pinMode(txPin, OUTPUT);
      pinMode(rxPin, INPUT);

      data[0] = 'A';   // Set car timeout
      data[1] = 'T';
      data[2] = ' ';
      data[3] = 'S';
      data[4] = 'T';
      data[5] = ' ';
      data[6] = 'F';
      data[7] = '0';

      sp[0] = '0';     // Send req for speed
      sp[1] = '1';
      sp[2] = ' ';
      sp[3] = '0';
      sp[4] = 'D';

      rm[0] = '0';     // Send req for RPM
      rm[1] = '1';
      rm[2] = ' ';
      rm[3] = '0';
      rm[4] = 'C';

      im[0] = '0';     // Send req for manifold pressure
      im[1] = '1';
      im[2] = ' ';
      im[3] = '0';
      im[4] = 'B';
```

```
        it[0] = '0';     // send req for air temperature
        it[1] = '1';
        it[2] = ' ';
        it[3] = '0';
        it[4] = 'F';
}
void test()
{
   for (int i = 0; i<8; i++)
           {
             Serial.print(data[i]);
           }
           Serial.println();
           Serial.flush();
}

void Speed()
{
  for (int i = 0; i<5; i++)
           {
             Serial.print(sp[i]);
           }
           Serial.println();
           Serial.flush();

}
void rpm()
{
  for (int i = 0; i<5; i++)
           {
             Serial.print(rm[i]);
           }
           Serial.println();
           Serial.flush();

}
void imp()
{
  for (int i = 0; i<5; i++)
           {
             Serial.print(im[i]);
           }
           Serial.println();
           Serial.flush();

}
void iat()
{
  for (int i = 0; i<5; i++)
           {
             Serial.print(it[i]);
           }
           Serial.println();
           Serial.flush();

}
void readobd1(){
                for (i; i<3+p; i++){
                incomingByte = Serial.read();
                info[i] = incomingByte;
                }
                p=p+3;
                delay(50);
                Serial.println("Got OBD data");

}
void readobd2(){

                for (i; i<6+p; i++){
                incomingByte = Serial.read();
                info[i] = incomingByte;
                }
                p=p+6;
                delay(50);
                Serial.println("Got OBD data");

}
void Read()
{
for (int l = 0; l<99; l++)
 {
       Serial.print(info[l]);
 }

}
void loop() {
        delay(500);
    if (Serial.available() > 0 && readok != 4) {
         incomingByte = Serial.read();
         if (incomingByte == '>')
         {
           test();
           delay(500);
         }
        else if(incomingByte =='K')
        {
          if (readok == 0)
          {
            Speed();
            i = 0;
            p = 0;
          }
          else if (readok == 1)
          {
          rpm();
          }
          else if (readok == 2)
          {
          imp();
```

```
  }
  else if (readok == 3)
  {
  iat();
  }
}
else if(incomingByte =='4')
{
  incomingByte = Serial.read();

    if(incomingByte =='1')
    {
      incomingByte = Serial.read();
      incomingByte = Serial.read();
       if(incomingByte =='0')
          {
           incomingByte = Serial.read();
             if (incomingByte == 'D')
             {
               readobd1();
               readok = 1;
             }
             else if (incomingByte == 'C')
             {
               readobd2();
               readok = 2;
             }
             else if (incomingByte == 'B')
             {
               readobd1();
               readok = 3;
             }
             else if (incomingByte == 'F')
             {
               readobd1();
               readok = 4;
             }

          }

    }
}
else if(incomingByte =='Y')
{
Serial.println("Bus Busy");
}
else if(incomingByte =='<')
{
Serial.println("<Data Error");
}
else if(incomingByte =='D')
{
Serial.println("Data Error");
}
else if(incomingByte =='N')
{
Serial.println("No data");
          Read();
          }
          else if(incomingByte =='B')
          {
          Serial.println("Could be Buss Error");
          }


           }
          else while (readok == 4)
              {
           gpsByte = gps.read();
          if (gpsByte == 'R')
              {
                readok = 0;
                for (i; i<70+p; i++){
               gpsByte = gps.read();
               info[i] = gpsByte;
               delay(1);
               }
               p=p+70;
              Serial.println("GPS yea");
               Read();
               Wire.beginTransmission(4);
               Wire.send(info);            // Send data to SD card
               Wire.endTransmission();
               }
            }


}
```

Software Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Web;
using System.Net;
using System.IO;
using System.Configuration;
using System.Data.SqlClient;

namespace MOVE
{
   public partial class Main : Form
   {
      string[] IDS = new string[2000]; // Contains MPG
Values
      string[] IDS2 = new string[2000];// Contains Speed
Values
      string[] IDS6 = new string[2000];// Contains RPM
      double IDS3 = 0; //Contains Maximum Value for Speed
```

```
        int IDS4 = 0; //Contains Maximum Value for RPM
        double IDS5 = 0; //Contains Maximum Value for MPG
        string[] IDS_Lat = new string[2000]; //Contains
Latitude Value
        string[] IDS_Long = new string[2000]; //Contains
Longitude Value
        string ChartX_RPM = "";
        string ChartX_SPEED = "";
        string ChartY_MPG = "";
        int MaxX_RPM = 0;
        double MaxX_Speed = 0;
        double MaxY = 0;
        string Map_URL = "";
        Car Make = new Car();
        bool datacheck = false;


        public Main()
        {

            InitializeComponent();
            label3.Hide();
            label4.Hide();
            proBar.Hide();
            Make.Show();
            Make.TopMost = true;
            Make.Activate();
            //NewCar.Hide();
            //pbInfo.SendToBack();
           // lblMake.Text =
Make.Make.comboBox1.SelectedText;
        }

        private void splitContainer1_Panel2_Paint(object
sender, PaintEventArgs e)
        {

        }

        private void splitContainer1_Panel1_Paint(object
sender, PaintEventArgs e)
        {

        }

        private void aboutToolStripMenuItem1_Click(object
sender, EventArgs e)
        {

            About About = new About();
            About.Show();
        }

        private void exitToolStripMenuItem_Click(object
sender, EventArgs e)
        {
        }
        private void btnFuel_Click(object sender, EventArgs e)
        {
            if (Make.comboBox1.SelectedItem == null ||
Make.comboBox2.SelectedItem == null ||
Make.comboBox3.SelectedItem == null ||
Make.comboBox4.SelectedItem == null ||
Make.comboBox5.SelectedItem == null)
            {
                Make.SendToBack();
                MessageBox.Show("Please fill in all vehicle
information before proceeding.", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }
            else
            {
                string DataPoints_Speed = "";
                string DataPoints_RPM = "";
                //int j = 0;

                if (lstItems.Items.Count == 0)
                {
                    MessageBox.Show("Please run the Data first in
order to get a data set for graphing!", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
                }

                else
                {
                    /******************************** Query to
get Max Value for y-axis
*********************************/

                    string Conn =
ConfigurationManager.ConnectionStrings["SQL_Conn"].ToS
tring();

                    //SQL CONNECTION STRING
                    SqlConnection conn_Fuel = new
SqlConnection(Conn);

                    //SQL COMMAND TO SELECT ITEMS

                    SqlCommand DataCMD = new
SqlCommand("select max(MPG) from GPSInfo", conn_Fuel);

                    DataCMD.CommandTimeout = 30;

                    //OPEN SQL CONNECTION
                    conn_Fuel.Open();


                    //RUN QUERY
                    IDataReader iddr = DataCMD.ExecuteReader();

                    while (iddr.Read())
                    {
```

```
            //IDS3 = (int) iddr[0];
            string convert = (iddr[0].ToString());
            //IDS5 = Int32.Parse("convert");
            //IDS5 = Double.Parse("convert");
            IDS5 = Convert.ToDouble(iddr[0]);
        }

        conn_Fuel.Close();
        conn_Fuel.Dispose();
        /****************************** End of Query  *************************/

        /********************* Query to get Max Speed for x-axis *********************/
        string Conn2 = ConfigurationManager.ConnectionStrings["SQL_Conn"].ToString();

        //SQL CONNECTION STRING
        SqlConnection conn_Fuel2 = new SqlConnection(Conn2);

        //SQL COMMAND TO SELECT ITEMS

        SqlCommand DataCMD2 = new SqlCommand("select max(Speed) from GPSInfo", conn_Fuel2);
        DataCMD.CommandTimeout = 30;

        //OPEN SQL CONNECTION
        conn_Fuel2.Open();


        //RUN QUERY
        IDataReader iddr2 = DataCMD2.ExecuteReader();



        while (iddr2.Read())
        {
            //IDS4 = (int)iddr2[0];
            IDS3 = Convert.ToDouble(iddr2[0]);
        }

        conn_Fuel2.Close();
        conn_Fuel2.Dispose();

        /********************* End of Query ******************/

        /********************* Query to get Max RPM for x-axis *********************/
        string Conn3 = ConfigurationManager.ConnectionStrings["SQL_Conn"].ToString();

        //SQL CONNECTION STRING
        SqlConnection conn_Fuel3 = new SqlConnection(Conn3);

        //SQL COMMAND TO SELECT ITEMS

        SqlCommand DataCMD3 = new SqlCommand("select max(RPM) from GPSInfo", conn_Fuel3);
        DataCMD.CommandTimeout = 30;

        //OPEN SQL CONNECTION
        conn_Fuel3.Open();


        //RUN QUERY
        IDataReader iddr3 = DataCMD3.ExecuteReader();



        while (iddr3.Read())
        {
            //IDS4 = (int)iddr2[0];
            IDS4 = Convert.ToInt32(iddr3[0]);
        }

        conn_Fuel3.Close();
        conn_Fuel3.Dispose();

        /********************* End of Query ******************/


        //Encoding for the chart URL
        for (int i = 0; i < lstItems.Items.Count; i++)
        {

            if (i == lstItems.Items.Count - 1)
            {
                ChartX_RPM = ChartX_RPM + IDS6[i];
                ChartX_SPEED = ChartX_SPEED + IDS2[i];
                ChartY_MPG = ChartY_MPG + IDS[i];
            }
            else
            {
                ChartX_RPM = ChartX_RPM + IDS6[i] + ",";
                ChartX_SPEED = ChartX_SPEED + IDS2[i] + ",";
                ChartY_MPG = ChartY_MPG + IDS[i] + ",";
            }
        }

        //Finished String for the Chart Url
```

```
            //DataPoints_RPM = ChartX_RPM + "|" +
ChartY_MPG;//LINE CHART
            DataPoints_RPM = ChartX_RPM; // Bar Chart
            //DataPoints_Speed = ChartX_SPEED + "|" +
ChartY_MPG;//LINE CHART
            DataPoints_Speed = ChartX_SPEED;//LINE
CHART
            //Maximum Number on x-axis
            MaxX_RPM = IDS4;
            MaxX_Speed = IDS3;


            //Maximum Number on y-axis
            MaxY = IDS5;

            //Displays area where Chart will go
            pbChart.Show();
            btnRefresh.Show();

            try
            {
               /*****Using Google Charts...Pass data in
certain format and returns chart.******/

//pbChart.Load("http://chart.apis.google.com/chart?chs=500x
350&chd=t:" + DataPoints_RPM +
"&cht=lxy&chm=o,000000,0,-
1,5&chco=0000FF&chtt=RPM+vs+MPG&chxt=x,y&chxr=0,
0," + MaxX_RPM + "|1,0," + MaxY + "&chds=0," +
MaxX_RPM + ",0," + MaxY);//LINE CHART

//pbChart2.Load("http://chart.apis.google.com/chart?chs=500
x350&chd=t:" + DataPoints_Speed +
"&cht=lxy&chm=o,000000,0,-
1,5&chco=0000FF&chtt=Speed
(MPH)+vs+MPG&chxt=x,y&chxr=0,0," + MaxX_Speed +
"|1,0," + MaxY + "&chds=0," + MaxX_Speed + ",0," +
MaxY);//LINE CHART

pbChart.Load("http://chart.apis.google.com/chart?chs=500x3
50&chba=a&chd=t:" + DataPoints_RPM +
"&cht=bvg&chm=o,000000,0,-
1,5&chco=0000FF&chtt=RPM+vs+MPG&chxt=x,y&chxr=0,
0," + MaxX_RPM + "|1,0," + MaxY + "&chds=0," +
MaxX_RPM + ",0," + MaxY);//BAR CHART

pbChart2.Load("http://chart.apis.google.com/chart?chs=500x
350&chba=a&chd=t:" + DataPoints_Speed +
"&cht=bvg&chm=o,000000,0,-
1,5&chco=0000FF&chtt=Speed
(MPH)+vs+MPG&chxt=x,y&chxr=0,0," + MaxX_Speed +
"|1,0," + MaxY + "&chds=0," + MaxX_Speed + ",0," +
MaxY);//BAR CHART

            }
            catch
            {
                MessageBox.Show("Please check your
internet connection and try again.", "Check Connection",
MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
         }
      }
      private void btnRefresh_Click(object sender, EventArgs
e)
      {
         if (Make.comboBox1.SelectedItem == null ||
Make.comboBox2.SelectedItem == null ||
Make.comboBox3.SelectedItem == null ||
Make.comboBox4.SelectedItem == null ||
Make.comboBox5.SelectedItem == null)
         {
            Make.SendToBack();
            MessageBox.Show("Please fill in all vehicle
information before proceeding.", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
         }
         else
         {
            DialogResult Result = MessageBox.Show("Are you
going to continue analyzing the same car?", "Fuel Analyzer",
MessageBoxButtons.YesNoCancel,
MessageBoxIcon.Question);
            if (Result == DialogResult.Yes)
            {
               //MessageBox.Show("YES");
               pbChart.Image = null;
               pbChart2.Image = null;
               lstItems.Items.Clear();
               proBar.Value = 0;
               proBar.Hide();
               ChartX_RPM = "";
               ChartY_MPG = "";
            }
            else if (Result == DialogResult.No)
            {
               datacheck = false;
               MessageBox.Show("Please reconnect after
choosing new vehicle.","Fuel
Analyzer",MessageBoxButtons.OK,MessageBoxIcon.Informa
tion);
               Make.Refresh();
               Make.comboBox1.SelectedItem = "Chevrolet";
               Make.comboBox1.SelectedItem = "Acura";
               Make.Show();
               pbChart.Image = null;
               pbChart2.Image = null;
               lstItems.Items.Clear();
               proBar.Value = 0;
               proBar.Hide();
               ChartX_RPM = "";
               ChartY_MPG = "";
            }
```

```
        }
    }
    private void exitToolStripMenuItem1_Click(object
sender, EventArgs e)
    {
        Close();
    }

    private void btnGPS_Click(object sender, EventArgs e)
    {
        if (Make.comboBox1.SelectedItem == null ||
Make.comboBox2.SelectedItem == null ||
Make.comboBox3.SelectedItem == null ||
Make.comboBox4.SelectedItem == null ||
Make.comboBox5.SelectedItem == null)
        {
            Make.SendToBack();
            MessageBox.Show("Please fill in all vehicle
information before proceeding.", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        }
        else
        {
            if (lstItems.Items.Count == 0)
            {
                MessageBox.Show("Please run the Data first in
order to get a data set for graphing!", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }

            else if (rbMonthly.Checked == true ||
rbWeekly.Checked == true)
            {
                MessageBox.Show("Mapping is not available in
this mode. Please select Daily and refresh your data button.",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            else
            {
                //Encoding for the MAP URL
                for (int i = 0; i < lstItems.Items.Count; i++)
                {

                    if (i == lstItems.Items.Count - 1)
                    {
                        Map_URL = Map_URL + IDS_Lat[i] + ","
+ IDS_Long[i];

                    }
                    else
                    {
                        Map_URL = Map_URL + IDS_Lat[i] + ","
+ IDS_Long[i] + "|";
                    }
                }
                string MarkerStart = IDS_Lat[0] + "," +
IDS_Long[0];
                string MarkerEnd =
IDS_Lat[lstItems.Items.Count - 1] + "," +
IDS_Long[lstItems.Items.Count - 1];
                //Finished String for the Map Url is Map_URL

                /*Allows for us to open new form and pass the
URL for the path taken by user*/
                MapView Map = new MapView();
                Map.Show();
                Map.pbMap.ImageLocation = "center";

                try
                {

Map.pbMap.Load("http://maps.google.com/staticmap?size=5
75x370&maptype=hybrid&markers=" + MarkerStart+
",midgreens|"+MarkerEnd+",midrede&path=rgba:0x0000FF8
0,weight:5|" + Map_URL +
"&key=ABQIAAAAkYXfyw6elok5Bsw3NSAIJRT2yXp_ZA
Y8_ufC3CFXhHIE1NvwkxSsEAa8tmMPUxNwtU1U5RKL8
9uCEg");
                }
                catch
                {
                    MessageBox.Show("Please check your
internet connection and try again.", "Check Connection",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                    Map.Close();
                }
            }
        }
    }
    private void dataToolStripMenuItem_Click(object
sender, EventArgs e)
    {

    }

    private void Main_Load(object sender, EventArgs e)
    {
        // TODO: This line of code loads data into the
'dsGPS.GPSInfo' table. You can move, or remove it, as
needed.
        //this.gPSInfoTableAdapter.Fill(this.dsGPS.GPSInfo);

    }

    private void pbInfo_Click(object sender, EventArgs e)
    {

    }

    private void rbDaily_CheckedChanged(object sender,
EventArgs e)
```

```
        {
            Calendar cal = new Calendar();
            cal.Show();
        }

        private void btnConnect_Click(object sender, EventArgs
e)
        {
            if (datacheck == true)
            {
                MessageBox.Show("You are already connected.",
"Connect", MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }

            if (Make.comboBox1.SelectedItem == null ||
Make.comboBox2.SelectedItem == null ||
Make.comboBox3.SelectedItem == null ||
Make.comboBox4.SelectedItem == null ||
Make.comboBox5.SelectedItem == null)
            {
                Make.SendToBack();
                MessageBox.Show("Please fill in all vehicle
information before proceeding.", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }
            else
            {
                label3.Show();
                label4.Show();
                lblMake.Text = Make.labels.Make;
                lblModel.Text = Make.labels.Model;
                lblYear.Text = Make.labels.Year;
                lblcity.Text = Make.labels.City;
                lblhigh.Text = Make.labels.Highway;
                datacheck = true;
                Parse GetInfo = new Parse();
                DataTable dtAdd = new DataTable();

                Cursor.Current = Cursors.WaitCursor;


                dtAdd =
GetInfo.Convert("C:\\MOVETEXT\\data.txt");



                if (dtAdd == null)
                {
                    MessageBox.Show("Error processing");
                    return;
                }


                string Conn =
ConfigurationManager.ConnectionStrings["SQL_Conn"].ToS
tring();
                SqlConnection conn_File = new
SqlConnection(Conn);

                SqlBulkCopy bc = new SqlBulkCopy(Conn,
SqlBulkCopyOptions.CheckConstraints);
                bc.DestinationTableName = "GPSInfo";
                bc.BatchSize = dtAdd.Rows.Count;
                conn_File.Open();
                bc.WriteToServer(dtAdd);
                bc.Close();


            }
        }
        public void btnData_Click(object sender, EventArgs e)
        {
            if (Make.comboBox1.SelectedItem == null ||
Make.comboBox2.SelectedItem == null ||
Make.comboBox3.SelectedItem == null ||
Make.comboBox4.SelectedItem == null ||
Make.comboBox5.SelectedItem == null)
            {
                Make.SendToBack();
                MessageBox.Show("Please fill in all vehicle
information before proceeding.", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);



            }
            else if (datacheck == false)
            {
                MessageBox.Show("Please Connect", "Fuel
Analyzer",
MessageBoxButtons.OK,MessageBoxIcon.Information);
            }
            else
            {
                /*
                lblMake.Text = Make.labels.Make;
                lblModel.Text = Make.labels.Model;
                lblYear.Text = Make.labels.Year;
                lblcity.Text = Make.labels.City;
                lblhigh.Text = Make.labels.Highway;
                */


                int i = 0;
                int j = 0;
                proBar.Show();

                /**********************Query to fill
data*******************************/

                string Conn =
ConfigurationManager.ConnectionStrings["SQL_Conn"].ToS
tring();
                //Clear Items from List
                lstItems.Items.Clear();
```

```
        //SQL CONNECTION STRING
        SqlConnection conn_Fuel = new
SqlConnection(Conn);

        //SQL COMMAND TO SELECT ITEMS

        SqlCommand DataCMD = new
SqlCommand("SELECT * FROM Gpsinfo", conn_Fuel);
        DataCMD.CommandTimeout = 30;

        //OPEN SQL CONNECTION
        conn_Fuel.Open();


        //RUN QUERY
        IDataReader iddr = DataCMD.ExecuteReader();



        while (iddr.Read())
        {
           IDS[i] = iddr[0].ToString();
           IDS2[i] = iddr[1].ToString();
           IDS6[i] = iddr[2].ToString();
           IDS_Lat[i] = iddr[6].ToString();
           IDS_Long[i] = iddr[8].ToString();
           i++;
        }

        conn_Fuel.Close();
        conn_Fuel.Dispose();

        /********************** End of Query
*****************************************/

        //Sets value on Progress Bar
        proBar.Maximum = i - 1;

        //Places items in the List
        int counter = i;
        while (j < counter)
        {
           ListViewItem li = new ListViewItem(IDS[j]);
           li.SubItems.Add(IDS2[j]);
           li.SubItems.Add(IDS6[j]);
           lstItems.Items.Add(li);
           proBar.PerformStep();
           j++;
        }
        proBar.Value = 0;  //Resets Progess Bar
        proBar.Hide();     //Hides Progress Bar once
complete

      }
   }
```

```
      private void rbWeekly_CheckedChanged(object sender,
EventArgs e)
      {
         MessageBox.Show("There is not enough data.", "Fuel
Analyzer", MessageBoxButtons.OK);
      }

      private void rbMonthly_CheckedChanged(object sender,
EventArgs e)
      {
         MessageBox.Show("There is not enough data.", "Fuel
Analyzer", MessageBoxButtons.OK);
      }


   }
}
```

Parser

```
using System;//For strings and things
using System.IO;//to work with files
using System.Text.RegularExpressions;//to parse the file
using System.Data;//to generate a DataSet
using System.Data.SqlTypes;



namespace MOVE
{

   public class Parse
   {
      private static StreamReader stream;
      private static String val;

      public DataTable Convert(string sLogFile)
      {
         if(!System.IO.File.Exists(sLogFile))
            return null;//fail file not found

         //Create a new stream for the input file
         stream = new StreamReader(sLogFile);


         //Create a new data set
         DataSet dsFile = new DataSet();



         //Create a new data table
         DataTable dtTable = new DataTable();



         //Add columns
         dtTable.Columns.Add("MPG", typeof(float));
```

```
        dtTable.Columns.Add("Speed", typeof(float));
        dtTable.Columns.Add("RPM", typeof(string));
        dtTable.Columns.Add("Type", typeof(string));
        dtTable.Columns.Add("TimeStamp", typeof(int));
        dtTable.Columns.Add("Valid", typeof(char));
        dtTable.Columns.Add("Latitude", typeof(string));
        dtTable.Columns.Add("DirectionV", typeof(char));
        dtTable.Columns.Add("Longitude", typeof(string));
        dtTable.Columns.Add("DirectionH", typeof(char));
        dtTable.Columns.Add("Knots", typeof(string));
        dtTable.Columns.Add("TrueCourse", typeof(string));
        dtTable.Columns.Add("Date", typeof(int));
        dtTable.Columns.Add("Variation", typeof(string));
        dtTable.Columns.Add("Headed", typeof(string));
        dtTable.Columns.Add("Checksum", typeof(string));

        //Set Regular Expression
        Regex r = new
Regex(",(?=([^\"]*\"[^\"]*\")*(?![^\"]*\"))");



        // Get the first line of data from the input (log) file.
        val = NextLine(stream);
        //Strip out each field and insert in DataTable
        int sStart;//used as a placeholder for our current
location
        int nCount;//used to determine which field / column
we are currently extracting
        string sTemp;//used to temporarily hold the field
pulled out by RegEx in order to trim the quotation marks
from the ends of the fiel (if any)

        //Data row object used to set values
        DataRow drRow;

        //DataRow - adding data

        while(val != null)//val is empty when the input stream
has been read to the end
        {
            string[] Conversions = new string[9];
            Conversions[0] = val.Substring(0, 2); //Temp
holder for Speed (KM/H)
            Conversions[1] = val.Substring(3, 2); //Temp
holder for 1st RPM byte
            Conversions[2] = val.Substring(6, 2); //Temp
holder for 2nd RPM byte
            Conversions[3] = val.Substring(9, 2); //Temp
holder for Air Intake
            Conversions[4] = val.Substring(12, 2); //Temp
holder for Air Temperature
            Conversions[5] = val.Substring(15, 2); //Temp
holder for GPS sentence TYPE
            Conversions[6] = ""; //temp holder for speed MPH
            Conversions[7] = "";//temp for RPM
            Conversions[8] = ""; //temp for MPG

            int[] Converted = new int[5];
            double Speed_KMH =
System.Convert.ToInt32(Conversions[0], 16);
            double Speed_MPH = Speed_KMH * 0.621;

            Converted[0] =
System.Convert.ToInt32(Conversions[0], 16);  //Speed
(KM/H)
            Converted[1] =
System.Convert.ToInt32(Conversions[1], 16);  //RPM 1st
            Converted[2] =
System.Convert.ToInt32(Conversions[2], 16);  //RPM 2nd
            Converted[3] =
System.Convert.ToInt32(Conversions[3], 16);  //Air Intake
            Converted[4] =
System.Convert.ToInt32(Conversions[4], 16);  //Air Temp

            int RPM = ((Converted[1] * 256) + Converted[2]) /
4;

            int AirTemp = Converted[4] - 40;

            int MAF = RPM * (Converted[3] / AirTemp);


            double MPG = 4*(710.7 * (Speed_KMH /
System.Convert.ToInt32(MAF)));

            Conversions[6] = Speed_MPH.ToString(); // Speed
MPH

            Conversions[7] = RPM.ToString(); //RPM Reading
            Conversions[8] = MPG.ToString(); //MPG Reading
            //Conversions[8] = Conversions[8].Substring(0, 1);
            string rebuild = Conversions[8] + "," +
Conversions[6] + "," + Conversions[7] + "," + Conversions[5]
+ ",";
            string trim = val.Substring(18);
            string newval = rebuild + trim;
            val = newval;

           sStart=0;
           nCount=0;

           //create a new row
           drRow=dtTable.NewRow();

           //Iterate through all the matches (match=field in
current row)
           foreach (Match m in r.Matches(val))
           {
              //Retrieve the field based on the results of the
match
              sTemp=val.Substring(sStart, m.Index - sStart);
              //Get rid of pesky quotes around certain fields
              sTemp=sTemp.Trim("");
```

```
            //but that wouldn't make sense here
             drRow[nCount]=sTemp;

            //keep one step ahead in the matching game..
            sStart = m.Index + 1;

            //keep track of which field is next
            nCount++;
          }

          sTemp=val.Substring(sStart, val.Length - sStart);

          sTemp=sTemp.Trim('"');

           drRow[nCount]=sTemp;


          dtTable.Rows.Add(drRow);


          val = NextLine(stream);


        }

        //Return the DataTable
        return dtTable;


      }



      //Extrace one line of text at a time from an input stream
      private static String NextLine(StreamReader stream)
      {
        int stemp = stream.Read();


         string greturn = stream.ReadLine();

        return greturn;

      }

    }
}
```